Discrete Applied Mathematics 160 (2012) 604–617

Contents lists available at SciVerse ScienceDirect

Discrete Applied Mathematics

journal homepage: www.elsevier.com/locate/dam

# The universality of iterated hashing over variable-length strings

Daniel Lemire *

*LICEF, Université du Québec à Montréal (UQAM), 100 Sherbrooke West, Montreal, QC, H2X 3P2 Canada*

ARTICLE INFO

*Article history:*
Received 10 August 2010
Received in revised form 31 October 2011
Accepted 8 November 2011
Available online 3 December 2011

*Keywords:*
Iterated hashing
Hashing strings
Permutations
Finite fields

ABSTRACT

Iterated hash functions process strings recursively, one character at a time. At each iteration, they compute a new hash value from the preceding hash value and the next character. We prove that iterated hashing can be pairwise independent, but never 3-wise independent. We show that it can be almost universal over strings much longer than the number of hash values; we bound the maximal string length given the collision probability.



## 1. Introduction

We consider hash functions mapping variable-length strings to $L$-bit integers. They have numerous applications from indexing – e.g., with hash tables [9,20,3,33] and Bloom filters [24] – to spell-checking [13], compression [17] and cryptography [26].

We consider hash functions $h$ picked randomly from a family $\mathcal{H}$ [4]. We focus on *iterated hash functions* [23,15]: given a string $s_1 s_2 \cdots s_n$, starting from an initial value (or seed) $H_0$, the hash value of the whole string, $H_n$, is computed recursively from a *compression function* $F$ as $H_i = F(H_{i-1}, s_i)$ for $i = 1, 2, \ldots, n$. Thus, a hash function is defined both by an initial value $H_0$ and a compression function $F$. A typical example is Carter–Wegman Polynomial Hashing over finite fields [12,4], that is, hash functions of the form $h(s) = \sum_i t^{n-i} s_i$ for some randomly chosen element $t$. Many hash functions over variable-length strings are iterated including Pearson hashing [21,32], SAX and SXX [25] as well as the hash functions commonly used in C++ and Java. In cryptography, iterated hashing is also known as Merkle–Damgård [16] hashing; it includes the popular functions MD4, MD5, SHA-0 and SHA-1.

Good hash functions are such that hash values *appear* random. Formally, a family is (pairwise) *universal* if the probability of a collision is no larger than if the hash values were random: $P\left(h(s) = h(s')\right) \leq 1/2^L$ for $s \neq s'$. It is $\varepsilon$-almost universal [31] (or $\varepsilon$-AU) if the probability of a collision is bounded by $\varepsilon < 1$. Furthermore, a family is $k$-wise independent if given $k$ distinct elements $s^{(1)}, s^{(2)}, \ldots, s^{(k)}$, their hash values are independent:

$$P\left(h(s^{(1)}) = y^{(1)} \wedge h(s^{(2)}) = y^{(2)} \wedge \cdots \wedge h(s^{(k)}) = y^{(k)}\right) = \frac{1}{2^{kL}}$$

for any hash values $y^{(1)}, y^{(2)}, \ldots, y^{(k)}$. Pairwise (or 2-wise) independence implies pairwise universality and thus, it is also called strong universality.

* Tel.: +1 514 987 3000x2835; fax: +1 514 843 2160.
*E-mail addresses:* lemire@acm.org, lemire@gmail.com.

0166-218X/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.dam.2011.11.009

**Table 1**
Upper bounds on the string length for arbitrarily large iterated hash families. We write $LCM_{2^L}$ for the least common multiple of the integers from 1 to $2^L$.

| | Cardinality-based bounds (Section 6) | New bounds (Section 7) |
|---|---|---|
| Strongly universal | $L + 2\log 2^L! - \log(2^L - 1) - 1$ | $2^L + 1$ |
| Universal | $2L + L2^{L+1}$ | $2^L + 1$ |
| Almost universal (any $\varepsilon < 1$) | $L(2^{L(2^{L+1}+1)} + 1)$ | $2^L + LCM_{2^L} - 1$ |

The main contributions are as follows.

- We show that iterated hashing cannot be 3-wise independent (Section 3). Thus, we have to be satisfied with pairwise independence. We can get 3-wise independence if we consider a generalization of iterated hashing where there is a new compression function with each iteration in the computation. However, 4-wise independence remains impossible.
- We show that pairwise independence is possible for iterated hashing by presenting the TABULATED family (Section 5.2).
- We show that almost universality is possible for strings longer than $2^L$ characters, e.g., with the Pearson family (Section 5.4).
- Iterated hashing families have limited cardinality: there are only so many possible compression functions. This limits their universality. We apply results from [18,31] to derive new bounds (Section 6). To make one such bound tighter, we use the fact that pairwise independent families must have permuting compression functions, a concept we introduce in Section 4.
- We can derive tighter bounds using the innate limitations of iterated hashing (Section 7). Table 1 summarizes some of our results.

## 2. Preliminaries

We want $L$-bit integer hash values: hash functions map elements to integers in $\{0, 1, \ldots, 2^L - 1\}$. The weakest property we require from a family is uniformity: all hash values are equiprobable. That is a family is uniform if $P(h(s) = y) = \frac{1}{2^L}$ for any $s$ and $y$. (We pick $h$ uniformly at random from the family $\mathcal{H}$.) It is not difficult to construct such a family. For example, let $\mathcal{H}$ be the set of all $2^L$ distinct constant functions ($h(s) = c$ for all $s$). This family is uniform but a poor choice in practice because any two elements $s$ and $s'$ are sure to collide: $P\left(h(s) = h(s')\right) = 1$.

Thus, we commonly seek families satisfying stronger conditions. A family is $\varepsilon$-almost universal if the probability of a collision is bounded by $\varepsilon < 1$: $P\left(h(s) = h(s')\right) \leq \varepsilon$ for any $s$ and $s'$. We say that the family is universal when it is $1/2^L$-almost universal.

While bounding the probability of a collision is sufficient for some applications like conventional hash tables, other results require stronger properties. A family is pairwise independent (or strongly universal) if the hash values of any two elements are independent:

$$P\left(h(s) = y \wedge h(s') = y'\right) = \frac{1}{2^{2L}}$$

for any two distinct elements $s, s'$ and any two hash values $y, y'$. It is 3-wise independent if the hash values of any three elements are independent:

$$P\left(h(s) = y \wedge h(s') = y' \wedge h(s'') = y''\right) = \frac{1}{2^{3L}}$$

for any three distinct elements $s, s', s''$ and any three hash values $y, y', y''$. We can generalize this definition to $k$-wise independence. A family which is $k$-wise independent for any $k$ is fully independent. (A family is trivially $k$-wise independent over a set containing less than $k$ distinct elements. In our work, we implicitly assume that there are at least $k$ distinct elements whenever we consider $k$-wise independence.)

We have that $k$-wise independence implies $k - 1$-wise independence for $k \geq 2$. For example, suppose we have 3-wise independence. Then we have that

$$\begin{aligned} P\left(h(s) = y \wedge h(s') = y'\right) &= \sum_{y''=0}^{2^L-1} P\left(h(s) = y \wedge h(s') = y' \wedge h(s'') = y''\right) \\ &= 2^L \times \frac{1}{2^{3L}} = \frac{1}{2^{2L}}. \end{aligned}$$

Similarly, $k$-wise independence for any $k > 1$ implies uniformity.

A family that is universal, but not strongly universal might be *XOR universal* if the bitwise exclusive OR of hash values appears random, that is $P(h(s) \oplus h(s') = y) = 1/2^L$ for all distinct elements $s, s'$ and hash values $y$ [18,11]. (The symbol $\oplus$ is the bitwise exclusive OR.)

We can weaken both strong universality and XOR universality.

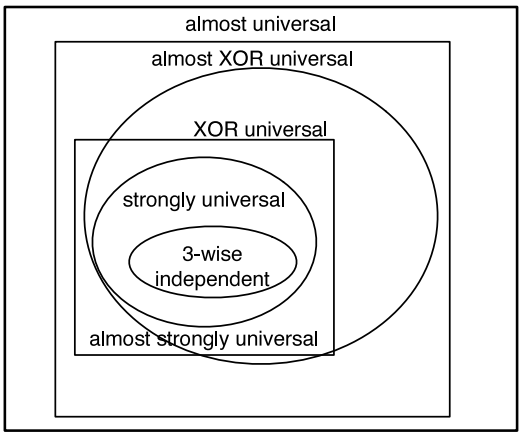


**Fig. 1.** Visual summary of the main properties related to universality. Strong universality is synonymous with pairwise independence.

- Given $\varepsilon < 1$, a family is $\varepsilon$-almost strongly universal if it is uniform and if $P(h(s) = y \wedge h(s') = y') \leq \varepsilon/2^L$ for any two distinct elements $s, s'$ and any two hash values $y, y'$. A $1/2^L$-almost strongly universal family is strongly universal.
- Similarly, it is $\varepsilon$-almost XOR universal (or $\varepsilon$-AXU) if the probability $P(h(s) \oplus h(s') = y)$ is bounded by $\varepsilon$.

We have that $\varepsilon$-almost strong universality implies $\varepsilon$-almost XOR universality which itself implies $\varepsilon$-almost universality. (See Fig. 1.)

We can also generalize almost universality to $k$-wise almost universality: a family is $k$-wise $\varepsilon$-almost universal [18] if the probability of a $k$-way collision is bounded by $\varepsilon$ for some $\varepsilon < 1$. That is, if we have $P(h(s^{(1)}) = h(s^{(2)}) = \cdots = h(s^{(k)})) \leq \varepsilon$ as long as the $k$ elements $s^{(1)}, s^{(2)}, \ldots, s^{(k)}$ are distinct. We have that $k$-wise $\varepsilon$-almost universality implies $k+1$-wise $\varepsilon$-almost universality. E.g., almost universality implies 3-wise almost universality.

## 3. Iterated hashing is pairwise independent at best

We write the concatenation $\mathtt{ab}$ of two strings $\mathtt{a}$ and $\mathtt{b}$ as $\mathtt{a} \parallel \mathtt{b} = \mathtt{ab}$. If $\emptyset$ is the empty string, then $\emptyset \parallel \mathtt{a} = \mathtt{a}$. We begin by characterizing iterated functions.

**Proposition 1.** *Consider hash functions over all strings, including the empty string. The following statements are equivalent.*

- *$\mathcal{H}$ is a family of iterated hash functions.*
- *For any $h \in \mathcal{H}$, whenever $h(s) = h(s')$ for a pair of strings $s, s'$, then $h(s \parallel s'') = h(s' \parallel s'')$ for any string $s''$.*

**Proof.** By induction, iterated hash functions satisfy the second point. Indeed, suppose that $h \in \mathcal{H}$ is an iterated function with a corresponding compression function $F$. Let $s''_i$ be the $i$th character of the string $s''_i$. By appending the first character of $s''$ to both strings ($s$ and $s'$) we get a collision: $h(s \parallel s''_1) = F(h(s), s''_1) = F(h(s'), s''_1) = h(s' \parallel s''_1)$. We can then append the remaining characters of $s''$ one by one starting with $s''_2$ and finally prove that $h(s \parallel s'') = h(s' \parallel s'')$.

Conversely, suppose the second point is true. Pick $h \in \mathcal{H}$. We want to construct a corresponding compression function $F$. For any hash value $y$ in the domain of $h$, there is at least one string $\rho_y$ such that $h(\rho_y) = y$. Let $F(y, \mathtt{a}) = h(\rho_y \parallel \mathtt{a})$ for all characters $\mathtt{a}$. By the second point, $F$ is well defined: its definition is independent of the choice of $\rho_y$. We can verify that the iterated hash function with compression $F$ and initial value $H_0 = h(\emptyset)$ agrees with $h$ on all strings which concludes the proof. □

Families of iterated hash functions have limited independence. The next lemma shows that they are pairwise independent at best. Moreover, almost strong universality (and thus strong universality) requires a non-fixed initial value.

**Lemma 1.** *Iterated hashing cannot be* 3*-wise independent, unless we bound the string length to two characters. Moreover, almost strong universality is impossible with a fixed initial value unless we bound the string length to one character.*

**Proof.** We prove the first statement by contradiction. Suppose that an iterated family $\mathcal{H}$ is 3-wise independent. By definition, we must have

$$P\left(h(\mathtt{a}) = y \wedge h(\mathtt{ab}) = y \wedge h(\mathtt{abb}) = y\right) = \frac{1}{2^{3L}}$$

for any hash values $y$, and any characters $\mathtt{a}$ and $\mathtt{b}$. (We allow $\mathtt{a} = \mathtt{b}$.) However, the family is also pairwise independent so that $P\left(h(\mathtt{a}) = y \wedge h(\mathtt{ab}) = y\right) = \frac{1}{2^{2L}}$. However, if $h(\mathtt{a}) = y$ and $h(\mathtt{ab}) = y$ then the compression function satisfies $F(y, \mathtt{b}) = y$

and therefore $h(\mathtt{abb}) = y$. Hence we conclude that

$$\begin{aligned}\frac{1}{2^{2L}} &= P\left(h(\mathtt{a}) = y \wedge h(\mathtt{ab}) = y\right)\\ &= P\left(h(\mathtt{a}) = y \wedge h(\mathtt{ab}) = y \wedge h(\mathtt{abb}) = y\right)\\ &= \frac{1}{2^{3L}},\end{aligned}$$

a contradiction when $L \geq 1$.

For the second statement, suppose that the family is $\varepsilon$-almost universal for $\varepsilon < 1$. Let the fixed initial value be $H_0$. If the family is pairwise independent then $P(h(\mathtt{a}) = H_0 \wedge h(\mathtt{aa}) = H_0) \leq \varepsilon/2^L$. Moreover, because almost strong universality implies uniformity, we have that $P(h(\mathtt{a}) = H_0) = 1/2^L$. Because $h$ is iterated, we have that $h(\mathtt{a}) = H_0$ implies that the compression function satisfies $F(H_0, \mathtt{a}) = H_0$. Hence, we have that $h(\mathtt{a}) = H_0$ implies $h(\mathtt{aa}) = H_0$. It follows that $P(h(\mathtt{a}) = H_0 \wedge h(\mathtt{aa}) = H_0) = P(h(\mathtt{a}) = H_0) = 1/2^L$ and therefore the family cannot be pairwise independent because $1/2^L > \varepsilon/2^L$. □

To allow better independence, we consider *generalized iterated hash functions* where a new compression function is used for each new character: $H_i = F_i(H_{i-1}, s_i)$ for $i = 1, 2, \ldots, n$. A family of generalized hash functions is such that whenever $h(s) = h(s')$ for a pair of strings $s$, $s'$ having the same length, then $h(s \parallel s'') = h(s' \parallel s'')$ for any string $s''$.

It includes hashing by multilinear functions over finite fields [4,28], e.g., hash functions of the form $h(s) = m_1 + \sum_i m_{i+1}s_i$ with randomly generated values $m_1, m_2, \ldots$ (henceforth MULTILINEAR). The compression functions are $F_i(y, c) = y + m_{i+1}c$ with an initial value of $m_1$. The computation is in the finite field $\mathbb{F}_p$: characters are mapped to elements of $\mathbb{F}_p$. For example, we can choose the field of cardinality $p = 2^L$: the polynomials with binary coefficients modulo $p(x)$, where $p(x)$ is an irreducible polynomial of degree $L$. We write $\mathbb{F}_{2^L} = \mathrm{GF}(2)[x]/p(x)$. That is, integers in $[0, 2^L)$ are represented as polynomials of degree $L-1$ having binary coefficients. Addition or subtraction is the bitwise (or term-wise) exclusive OR. Multiplication by $x$ is just the left shift, unless the left-most bit is 1, in which case the left shift must be followed by the addition with $p(x)$. Exhaustive lists of irreducible polynomials are available online [27]. Otherwise, when $p$ is a prime number, we merely have to compute $F_i(y, c) = m_{i+1}y + c \mod p$ using the usual integer algebra. We might prefer finite fields that have prime cardinality close to $2^L$. For example, we can set $p$ to some Mersenne prime such as $2^{17} - 1$, $2^{31} - 1$ or $2^{61} - 1$, or other convenient prime such as $2^{32} - 5$ or $2^{64} - 59$ [12].

**Lemma 2.** MULTILINEAR *is pairwise independent if we forbid strings ending with the value zero.*

**Proof.** We have that $h(\mathtt{a0}) = h(\mathtt{a})$ so universality is impossible if we allow strings to end with the value zero. So let $s$, $s'$ be two distinct strings of lengths $|s|$ and $|s'|$ ending with non-zero values. Assume without loss of generality that $|s| \geq |s'|$. Given $h(s') = y'$, we can solve for $m_1$ as a function of $y'$, $s'$ and the values $m_2, m_3, \ldots, m_{|s'|}$.

If $s$ is longer than $s'$ (that is $|s| > |s'|$), then we can solve for $m_{|s|}$ in $h(s) = y$ as a function of $y$, $s$ and $m_1, m_3, \ldots, m_{|s|-1}$. In turn, if we substitute our solution for $m_1$, we have $m_{|s|}$ as a function of $y$, $y'$, $s$, $s'$, and all $m_i$ for $i \neq 1$, $|s|$.

If $s = s'$, then there must be some $j$ such that $s_j \neq s'_j$. Hence, we can solve for $m_j$ in $h(s) - h(s') = y - y'$ as a function of $y$, $y'$, $s$, $s'$ and all $m_i$ for $i \neq 1, j$ after substituting the solution for $m_1$ from $h(s') = y'$.

Thus, in either case, among all possibles values of $m_1, m_2, \ldots, m_{|s|}$, two values are fixed by $h(s) = y \wedge h(s') = y'$. Hence, we have that $P(h(s) = y \wedge h(s') = y') = p^{|s|-2}/p^{|s|} = 1/p^2$ where $p$ is the cardinality of the field. This proves pairwise independence. □

If we choose zero as an initial value $m_1 = 0$, then MULTILINEAR is still XOR universal when $p = 2^L$. However, it fails to be pairwise independent. Indeed, given $\mathtt{a}$, and $\mathtt{b}$ two distinct elements of $\mathbb{F}_p$, we cannot satisfy both $h(\mathtt{a}) = m_1 + m_2\mathtt{a} = \mathtt{a}$ and $h(\mathtt{b}) = m_1 + m_2\mathtt{b} = \mathtt{a}$ unless $m_1 \neq 0$.

MULTILINEAR has a nearly optimal memory-universality trade-off. Indeed, Stinson [31] showed that pairwise independent families must have cardinality at least $1 + a(b - 1)$ where $a$ is the number of strings and $b$ is the number of hash values. There are $p^n - 1$ strings of length at most $n$ in $\mathbb{F}_p$ ending with a non-zero value. Thus, any pairwise independent family from strings in $\mathbb{F}_p$ of length bounded by $n$ to elements in $\mathbb{F}_p$ must have at least $1 + (p^n - 1)(p - 1)$ hash functions. That is, its cardinality is in $\Omega(p^{n+1})$. Meanwhile, there are $p^{n+1}$ different hash functions in MULTILINEAR when strings have length at most $n$.

We can have better universality than MULTILINEAR, at the cost of a higher memory usage. Consider sequences of 3-wise independent hash functions $h_i$ from characters to $L$-bit integers. The Zobrist [36,37] family of string hash functions $h(s) = h_1(s_1) \oplus h_2(s_2) \oplus \ldots \oplus h_n(s_n)$ is 3-wise independent [14,34,4]. It is also an example of generalized iterated hashing with the compression function $F_i(y, c) = y \oplus h_i(c)$. Moreover, it has optimal independence since generalized iterated families are 3-wise independent at best according to the next lemma.

**Lemma 3.** *Generalized iterated hashing cannot be* 4*-wise independent unless we bound the string length to one character.*

**Proof.** Consider any generalized iterated hash function $h$. If $h(s) = h(s')$ for any two strings $s$ and $s'$ of the same length then $h(s \parallel \mathtt{a}) = h(s' \parallel \mathtt{a})$ for any character $\mathtt{a}$. Hence, assuming that the family is 4-wise independent, we have that

$$\begin{aligned}\frac{1}{2^{4L}} &= P\left(h(s) = y \wedge h(s') = y \wedge h(s \parallel \mathtt{a}) = z \wedge h(s' \parallel \mathtt{a}) = z'\right)\\ &= 0\end{aligned}$$

whenever $z \neq z'$, a contradiction. □

In the following sections, we consider solely conventional iterated hashing.

## 4. Pairwise independence requires permuting compression functions

Permuting compression functions $F$ are such that $y \mapsto F(y, c)$ is a permutation of the hash values $y \in [0, 2^L)$ for any character $c$ and any compression function. Hence, if $y \neq y'$ then $F(y, c) \neq F(y', c)$ when $F$ is permuting.

(It is not necessary for $F$ to permute all integer values in $[0, 2^L)$. For example, the hash function mapping all strings to a constant ($h(s) = z$) has a corresponding compression function $F$ which is defined only as $F(z, c) = z$ for all characters $c$. It is trivially permuting over a single hash value $z$.)

We have that XOR universality or pairwise independence implies a fixed collision probability of $1/2^L$ between distinct strings. This, in turn, implies permuting compression functions by the next lemma.

**Lemma 4.** *An iterated hash family with fixed collision probability ($P(h(s) = h(s')) = \varepsilon$ for $s \neq s'$) over strings of length two or more has* permuting *compression functions.*

**Proof.** Consider two distinct strings $s$, $s'$. Consider any iterated hash family with fixed collision probability $\varepsilon$. We have

$$\begin{aligned}\varepsilon &= P(h(s \parallel \mathtt{c}) = h(s' \parallel \mathtt{c}))\\ &= P(h(s) = h(s')) + P(h(s) \neq h(s') \wedge h(s \parallel \mathtt{c}) = h(s' \parallel \mathtt{c}))\\ &= \varepsilon + P(h(s) \neq h(s') \wedge h(s \parallel \mathtt{c}) = h(s' \parallel \mathtt{c})).\end{aligned}$$

Thus, we have $h(s) \neq h(s') \Rightarrow h(s \parallel \mathtt{c}) \neq h(s' \parallel \mathtt{c})$ which proves the result. □

Consider the consequences of this lemma for $L = 1$. Over $\{0, 1\}$, there are only two permutations: the identity and an exchange of the two values (0 and 1). Hence, we have that $F(F(y, \mathtt{a}), \mathtt{b}) = F(F(y, \mathtt{b}), \mathtt{a})$ if $F$ is permuting. Therefore, the strings $\mathtt{ab}$ and $\mathtt{ba}$ always collide ($h(\mathtt{ab}) = h(\mathtt{ba})$). Thus – in general – XOR universality or pairwise independence over strings longer than $L$ characters is impossible.

However, permuting compression functions have benefits on their own, beside being a consequence of pairwise independence. Consider any fixed permuting compression function $F$ and any string $s$. Consider any two distinct initial values $H_0$ and $H_0'$. Then the hash value of $s$ computed with $H_0$ must differ from the hash value computed with $H_0'$ by induction on the number of characters in the string $s$. Thus, we have the following result. It holds true for strings of arbitrary length.

**Lemma 5.** *An iterated hash family with permuting compression functions and independently chosen equiprobable initial values is uniform.*

A compression function is *strongly permuting* if it is permuting and if $F(y, c) = F(y, c')$ implies $c = c'$. Strong permutation means that strings having a Hamming distance of one never collide. Of course, this precludes pairwise independence.

**Lemma 6.** *Given a strongly permuting iterated hash family, two strings differing by exactly one character never collide.*

## 5. Iterated hash families over variable-length strings

We are interested in hashing variable-length strings using iterated hash functions. Let $\Sigma$ be the set of all characters from which the strings are constructed; the number of distinct characters is $|\Sigma|$. We present a range of iterated families (see Tables 2 and 3). Other hash families appear in Appendix A.

### 5.1. Carter–Wegman polynomial hashing

Carter and Wegman [4] defined an almost universal family of iterated hash functions (henceforth CWPoly) using polynomials over a finite field $\mathbb{F}_p$ as $h(s) = \sum_i t^{n-i} s_i$ for $t \in \mathbb{F}_p$ randomly chosen and where $s_i$ is the $i$th character of the string $s$. In other words, we use the compression function $F(y, c) = ty + c$. Characters are interpreted as elements of $\mathbb{F}_p$. We are especially interested in binary fields where $p = 2^L$.

Unfortunately, CWPoly with an initial value of zero is such that $h(00) = h(0) = 0$. To fix this problem, we need to choose a non-zero initial value [12] such as 1. Thus, we have $h(00) = t^2$, $h(0) = t$ and $h(223) = t^3 + 2t^2 + 2t + 3$. Therefore, given

**Table 2**
Universality of some iterated families: $n$ is the maximal string length.

| Family | Universality |
|---|---|
| CWPoly | $n/2^L$-almost XOR universal |
| Tabulated | Pairwise independent for $n \leq L$ |
| ShiftTabulated | Pairwise independent on last $L-n+1$ bits |
| Pearson on unary strings | $\max_{i<n} d(i)/2^L$-almost universal |

**Table 3**
Computational complexity (per character) and memory usage in bits of some iterated families. There are $|\Sigma|$ characters in the alphabet.

| Families | Complexity | Memory usage |
|---|---|---|
| CWPoly | $O(L \log L 2^{O(\log^* L)})$ | $L$ |
| Tabulated and ShiftTabulated | $O(L)$ | $\|\Sigma\|L$ |
| Pearson | $O(L)$ | $\log(2^L!) \leq L2^L$ |

two strings $s$ and $s'$, $h(s) - h(s')$ is a non-zero polynomial of degree at most $\max(|s|, |s'|)$ where $|s|$ and $|s'|$ are the lengths of strings $s$ and $s'$. By the Fundamental Theorem of Algebra, such a polynomial has at most $\max(|s|, |s'|)$ solutions. Thus, the probability of collision between two strings of length at most $n$ is at most $\frac{n}{p}$. This probability bound is tight. Indeed, consider the polynomial of degree $n$ over $\mathbb{F}_p$, $\tau(t) = \prod_{i=0}^{n-1}(t-i)$. E.g., $\tau(t) = t^3 + 2t$ for $n = 3$ in $\mathbb{F}_3$. It has the $n$ distinct roots 0, 1, …, $n-1$. If $s$ is the character 0 repeated $n$ times, and $s'$ is the string corresponding to the coefficients of the polynomial $\tau(t)$, we have that $h(s) - h(s') = \tau(t)$, hence $P(h(s) = h(s')) = \frac{n}{2^L}$. Moreover, Nguyen and Roscoe show that CWPoly has optimal universality given the size of its family [18]. Sadly, CWPoly is not uniform, but the probability of any hash value $y$ is bounded: $P(h(s) = y) \leq \frac{n}{2^L}$ for any string $s$ of length $n$.

When the field size is a power of two ($p = 2^L$), then CWPoly is $n/2^L$-almost XOR universal. Indeed, we have that $P(h(s) \oplus h(s') = y)$ (for any $y$) is given by the probability that $h(s) + h(s') = y$ as polynomials in $\mathrm{GF}(2)[x]/p(x)$. Yet $h(s) + h(s')$ is a non-zero polynomial of degree at most $\max(|s|, |s'|)$ in $t$ and the result follows again by the Fundamental Theorem of Algebra.

Unfortunately, CWPoly cannot be almost strongly universal over variable-length strings even if we use a random initial value. Indeed, consider the equations $h(\mathtt{aa}) = \mathtt{a}$ and $h(\mathtt{a}) = 0$. They can be written explicitly as $H_0 t^2 + \mathtt{a}t + \mathtt{a} = \mathtt{a}$ and $H_0 t + \mathtt{a} = 0$ where $H_0$ is the initial value. We see that $h(\mathtt{a}) = 0$ implies $h(\mathtt{aa}) = \mathtt{a}$. Therefore, if CWPoly was $\varepsilon$-almost strongly universal for $\varepsilon < 1$, we would have that $\varepsilon/2^L \geq P(h(\mathtt{aa}) = \mathtt{a} \wedge h(\mathtt{a}) = 0) = P(h(\mathtt{a}) = 0) = 1/2^L$, a contradiction. It is still possible to modify CWPoly so that it becomes almost strongly universal. Unfortunately, the result is not an iterated family by our definition. To get this stronger property, we append to each string a random character before hashing. We state the general result over $\mathbb{F}_p$, but it obviously applies when $p = 2^L$.

**Lemma 7.** *Consider the family* CWPoly *with the parameter $t$ chosen randomly among the non-zero values of $\mathbb{F}_p$ and an initial value of 1. Moreover, we choose a random value $\zeta$ in $\mathbb{F}_p$ and append it to all strings before hashing them:*

$$h(s) = t^{|s|+1} + \sum_{i=1}^{|s|} t^i s_i + \zeta.$$

*If we consider strings of length at most $n$, then this modified* CWPoly *is $(n+1)/(p-1)$-almost strongly universal.*

**Proof.** With the random parameter $\zeta$, CWPoly is clearly uniform.

It remains to show that

$$P(h(s) = y \wedge h(s') = y') \leq \frac{n+1}{(p-1)p}$$

for any two distinct strings $s$, $s'$ and any hash values $y$, $y'$. We have that $h(s) - h(s') - y - y'$ is a non-zero polynomial of degree at most $n+1$. (E.g., if $s = \mathtt{ab}$ and $s' = \mathtt{cd}$ then $h(s) - h(s') - y - y' = t^2(\mathtt{a} - \mathtt{c}) + t(\mathtt{b} - \mathtt{d}) - y - y'$.) To see why it must be a non-zero polynomial, consider two cases. If $s$ and $s'$ have the same length, let $i$ be such that $s_i \neq s'_i$ then $h(s) - h(s') - y - y'$ as a polynomial over the variable $t$ has $(s_i - s'_i)t^{i+1}$ as its $i+1^{\text{th}}$ term. If $s$ and $s'$ have different lengths, assume without loss of generality that $s$ is longer, then the $|s|+1^{\text{th}}$ term of the polynomial is $t^{|s|+1}$ and therefore the polynomial has degree $|s|+1$ and is non-zero. Hence, the polynomial has at most $n+1$ roots. Moreover, we have that $h(s) - h(s') - y - y'$ is independent from $\zeta$. Given any $t$ such that $h(s) - h(s') - y - y' = 0$ is satisfied, there is only one value $\zeta$ (dependent on $t$) such that $h(s) = y$. Thus there are at most $n+1$ pairs of values $t$, $\zeta$ such that $t \neq 0$, $h(s) - h(s') - y - y'$ and $h(s) = y$ are satisfied. Therefore, the probability that $h(s) = y$ and $h(s') = y'$ are both true is bounded by $\frac{n+1}{(p-1)p}$ because there are $p-1$ possible non-zero values for $t$ and $p$ possible values for $\zeta$. □

When $L$ bits fit in a processor register, the running time of the compression function $F$ may be considered independent of $L$. More formally, however, the multiplication between two $L$-bit integers required by the compression function is in $O(L \log L 2^{O(\log^* L)})$ [7].

### 5.2. Iterated string hashing by tabulation

Hashing by tabulation [34,5,4,14] has good universality, at the expense of the memory usage. We adapt this strategy to iterated hashing of variable-length strings.

Consider $\Gamma$, a randomly chosen function from characters in $\Sigma$ to $L$-bit hash values. There are $2^{L|\Sigma|}$ such functions. We consider integers as elements of $\mathbb{F}_{2^L}$, that is, as polynomials with binary coefficients in $\mathrm{GF}(2)[x]/p(x)$ where $p(x)$ is an arbitrarily chosen irreducible polynomial of degree $L$. Consider the family of iterated hash functions with compression functions of the form $F(y, c) = xy + \Gamma(c)$ and an initial value chosen randomly (henceforth Tabulated). The compression function is permuting. Tabulated is pairwise independent.

**Proposition 2.** Tabulated *is pairwise independent for strings no longer than L characters.*

**Proof.** Consider two distinct strings $s$, $s'$ no longer than $L$ characters. We want to show that $P(h(s) = y \wedge h(s') = y') = 1/2^{2L}$ for any hash values $y$, $y'$.

Consider the equation $h(s) = y$ or

$$H_0 x^{|s|} + \sum_{i=1}^{|s|} x^{|s|-i} \Gamma(s_i) = y$$

where $H_0$ is the initial value. We can solve for the initial value as a function of $y$ and $s$:

$$H_0 = x^{-|s|} \left( y - \sum_{i=1}^{|s|} x^{|s|-i} \Gamma(s_i) \right). \tag{1}$$

We solve for $\Gamma(s_j)$ for one value of $j$, in terms that do not depend on $H_0$. This will allow us to conclude the proof. We consider two cases.

- Suppose that the strings have the same length ($|s| = |s'|$). From $h(s) = y$ and $h(s') = y'$, we get that $h(s) - h(s') = y - y'$. We have that the equation $h(s) - h(s') = y - y'$ is independent from $H_0$ because the terms $H_0 x^{|s|}$ and $H_0 x^{|s'|}$ cancel out. Because the strings are distinct, there must be an index $j \in \{1, 2, \ldots, |s|\}$ such that $s_j \neq s'_j$. Let the character $s_j$ occur at indexes $r_1, r_2, \ldots, r_k$ in string $s$ (by definition $j \in \{r_1, r_2, \ldots, r_k\}$) and at indexes $r'_1, r'_2, \ldots, r'_l$ in string $s'$. If we let $q = \sum_m x^{|s|-r_m} - \sum_m x^{|s|-r'_m}$, the equation $h(s) - h(s') = y - y'$ can be written as $q\Gamma(s_j) = \lambda$ for some value $\lambda \in \mathbb{F}_{2^L}$ independent from $\Gamma(s_j)$ and $H_0$. Because $j$ is in $\{r_1, r_2, \ldots, r_k\}$ but not in $\{r'_1, r'_2, \ldots, r'_l\}$, and because all $|s| - r_m$'s and all $|s| - r'_m$'s are less than $L$, we have that $q \neq 0$.
- Suppose that the strings have different lengths. Without loss of generality, assume that $|s| > |s'|$. From $h(s) = y$ and $h(s') = y'$, we get that $(h(s) - y) - x^{|s|-|s'|}(h(s') - y') = 0$. The equation $(h(s) - y) - x^{|s|-|s'|}(h(s') - y') = 0$ is independent from $H_0$ because terms $H_0 x^{|s|}$ and $H_0 x^{|s'|} \times x^{|s|-|s'|}$ cancel out. Consider the character $s_{|s|}$ and seek all indexes where it appears: write these indexes $r_1, r_2, \ldots, r_k$ for string $s$ (by definition $|s| \in \{r_1, r_2, \ldots, r_k\}$) and $r'_1, r'_2, \ldots, r'_l$ for string $s'$. We have that $q = \sum_m x^{|s|-r_m} - \sum_m x^{|s|-r'_m}$ is non-zero because 0 is in $\{|s| - r_1, |s| - r_2, \ldots, |s| - r_k\}$ but not in $\{|s| - r'_1, |s| - r'_2, \ldots, |s| - r'_l\}$ since $r'_m \leq |s'| < |s|$ for all $m$'s, and because the $|s| - r_m$'s and the $|s| - r'_m$'s are less than $L$. For the rest of the proof, we set $j = 1$.

Hence we can solve for $\Gamma(s_j)$ as $\Gamma(s_j) = q^{-1}\lambda$ whether the two strings have the same length or not. Eq. (1) gives $H_0$ as a function of $\Gamma(s_i)$ for $i = \{1, 2, \ldots, |s|\}$. So, our formula for $H_0$ depends on $\Gamma(s_j)$, but we can substitute $\Gamma(s_j) = q^{-1}\lambda$ in this formula (Eq. (1)) to get an expression for $H_0$ which does not depend on $\Gamma(s_j)$. Thus, from the equations $h(s) = y$ and $h(s') = y'$, we get one and only one value for $H_0$ and $\Gamma(s_i)$ as a function of the other tabulated values and of $y$ and $y'$. Both values are chosen at random among $2^L$ values and thus the result is shown. □

For binary strings, Tabulated has nearly optimal memory-universality trade-off. Indeed, there are $2^{L+1} - 2$ binary strings of length at most $L$. Hence, any pairwise independent family over such binary strings must contain at least $1 + (2^{L+1} - 2)(2^L - 1)$ hash functions [31]. Therefore, its cardinality must be in $\Omega(2^{2L})$. Meanwhile, Tabulated has $2^{2L}$ hash functions over binary strings.

### 5.3. Iterated string hashing by shifted tabulation

While Tabulated is pairwise independent, it requires operations in finite fields of cardinality $2^L$. Thankfully some microprocessors have instructions for computations in such finite fields [8]. Yet it can be expensive on some computers, even with such instructions. Fortunately, pairwise independence is possible without finite fields [6].

The barrel or circular shift is the invertible operation by which all bits all shifted, except for the last ones, which are brought back at the beginning. For $L$-bit values, the barrel left shift by one can be written as $y \circlearrowleft 1 = (y \ll 1) \oplus (y \gg L-1)$ where $\ll$ and $\gg$ are the left and right shifts. E.g., 11001 becomes 10011. Barrel shifting can be implemented efficiently in hardware [1]. The popular x86 and ARM instruction sets offer the `ror` instruction for this purpose.

Consider the hash family (henceforth ShiftTabulated) with compression functions of the form $F(y, c) = (y \circlearrowleft 1) \oplus \Gamma(c)$ where $\Gamma$ is a randomly chosen function from characters to $L$-bit hash values. (Whether we choose the barrel left or right shift is arbitrary.) We choose the initial value randomly. Because the compression function is permuting, ShiftTabulated is uniform.

The ShiftTabulated compression functions can be computed efficiently: one value to look-up, one barrel shift and one bitwise XOR. Moreover, ShiftTabulated can be described using the same compression function as Tabulated—$F(y, c) = xy + \Gamma(c)$—in GF(2)[$x$]/($x^L + 1$). (The polynomial $x^L + 1$ fails to be irreducible and thus, GF(2)[$x$]/($x^L + 1$) is merely a ring.)

The proof of the pairwise independence of Tabulated (see Proposition 2) relies on the fact that any non-zero element of the field GF(2)[$x$]/$p(x)$ is invertible, which includes any non-zero polynomial of degree at most $L-1$ from GF(2)[$x$]. In turn, this means that given any polynomial $q$ of degree at most $n-1 < L$, and any element $r$ of the field, we have

$$P(q\Gamma(s_i) = r) = 2^{-L}$$

whenever $\Gamma(s_i)$ is picked at random in the field, because the equation is only true when $\Gamma(s_i) = q^{-1}r$. The same is *almost* true in GF(2)[$x$]/($x^L + 1$). We write that two values are equal modulo the first $n-1$ bits if we ignore the first $n-1$ bits in the comparison. By Corollary 1 from [14], we have that

$$P(q\Gamma(s_i) = r \bmod \text{first } n-1 \text{ bits}) = 2^{-L+n-1}.$$

Hence, by a proof similar to Proposition 2, we have the following result.

**Lemma 8.** ShiftTabulated *is pairwise independent on the last* $L - n + 1$ *bits for strings no longer than* $n$ *characters.*

### 5.4. Pearson hashing

We define Pearson hashing by the family of compression functions $F(y, c) = A_{y \oplus c}$ where $A$ is an array containing a permutation of the values in $\{0, 1, \ldots, 2^L - 1\}$ [21]. These compression functions are strongly permuting: $A_{y \oplus c} = A_{y' \oplus c}$ implies $y = y'$, $A_{y \oplus c} = A_{y \oplus c'}$ implies $c = c'$. We pick the initial value uniformly at random. Thus, Pearson is uniform. To our knowledge, the exact universality of Pearson remains unknown. (We know that it can never be strongly universal because its compression function is strongly permuting.)

For $L = 2$, Pearson is 5/6-almost universal for strings no longer than four. That is, it is universal for strings of length $2^L$ unlike CWPoly. Brute-force numerical investigations for large values of $L$ is difficult.

To simplify the analysis, we focus on unary strings: strings made of a single character (such as `aaaa`). The following result is an upper bound on the universality of Pearson over general strings.

**Proposition 3.** *Pearson is* $\varepsilon$*-almost universal over unary strings of length at most* $n$ *for* $\varepsilon = \max_{i<n} d(i)/2^L$ *where* $d(i)$ *is the divisor function—the number of positive integers dividing* $i$*.*

**Proof.** Consider strings made of the character `a`. Let $\pi$ be the permutation $A_{\cdot \oplus \mathtt{a}}$. Fix the initial value $H_0$. A collision between two unary strings of lengths $k, k' \leq n$ is equivalent to the equation $\pi^k H_0 = \pi^{k'} H_0 \Rightarrow \pi^l H_0 = H_0$ for $|k - k'| = l < n$. Consider any solution $\varpi$ of this equation ($\varpi^l H_0 = H_0$), then let $\sigma$ be the smallest integer such that $\varpi^\sigma H_0 = H_0$. We bound the number of solutions using the fact that $\sigma$ must divide $l$.

Given $\sigma$, there are $(2^L - 1)!$ solutions to the equation $\pi^\sigma H_0 = H_0$ subject to $\pi^i H_0 \neq H_0$ for $i < \sigma$. Indeed, we have that $\pi H_0$ can be any value, except $H_0$—thus we have $2^L - 1$ possibilities. We have that $\pi^2 H_0$ can be any value expect $H_0$ and $\pi H_0$, hence we have $2^L - 2$ possibilities. And so on, up to $\pi^\sigma H_0$ which is predetermined. At that point, we have enumerated $(2^L - 1)(2^L - 2) \cdots (2^L - \sigma + 1)$ possibilities. Each one of these possibilities define how the values $H_0, \pi H_0, \ldots, \pi^{\sigma-1} H_0$ are permuted. The other $2^L - \sigma$ values can be permuted to any available value, generating $(2^L - \sigma)!$ possibilities for $(2^L - 1)!$ possibilities.

Thus there is a total of $d(l)(2^L - 1)!$ solutions and $(2^L)!$ different permutations: the ratio is

$$\frac{d(l)(2^L - 1)!}{(2^L)!} = \frac{d(l)}{2^L}.$$

This is true for any initial value $H_0$. The string length difference $l$ ranges between 1 and $n-1$: we must keep the maximum value $\max_{i<n} d(i)/2^L$. □

The function $\max_{i<n} d(i)$ grows slowly (see Fig. 2). Formally, we have that $\max_{i<n} d(i) \in o(i^\varepsilon)$ for all $\varepsilon > 0$. For $\max_{i<n} d(i)$ to be equal to $2^L$ – so that Pearson is no longer almost universal over unary strings – we need $n$ to be larger than the least common multiple of the integers from 1 to $2^L$. Thus, Pearson can be almost universal over unary strings much longer than $2^L$ characters.

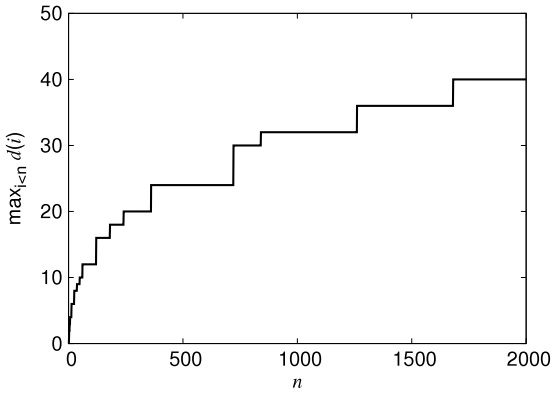


**Fig. 2.** Plot of $\max_{i<n} d(i)$ where $d$ is the divisor function.

**Table 4**
Numerically-derived upper bound on the collision probability between strings of length at most $n$ under Generalized Pearson ($L = 2$, $2^L = 4$).

| $n$ | Collision probability |
|---|---|
| 2 | 0.53 |
| 3 | 0.72 |
| $4 = 2^L$ | 0.84 |
| 5 | 0.88 |
| 6 | 0.89 |
| 7 | 0.95 |
| 8 | $\geq$0.97 |
| 9 | $\geq$0.98 |
| 10 | $\geq$0.99 |
| 11 | 1.00 |

To prove that iterated hashing for non-unary strings longer than $2^L$ characters is possible, we consider the following variation on Pearson hashing (henceforth Generalized Pearson): we pick compression functions of the form $F(y, c) = A_{y \oplus c}$ where $A$ is a random array containing values in $\{0, 1, \ldots, 2^L - 1\}$ (not necessarily a permutation). For $L = 1$, we have that $h(00) = h(11)$ with probability one. However, for $L = 2$, Generalized Pearson is almost universal for strings longer than $2^L$ (see Table 4). We computed these probabilities by enumerating all possible compression functions and all possible pairs of strings with $L$-bit characters.

## 6. Bounding the universality of iterated hashing by the maximal family size

Given only the number of hashable items, the number of hash values and the number of hash functions, we can bound the universality. To be $\varepsilon$-almost universal, a family must have enough hash functions.

There is a limited number of iterated hash functions. Compression functions have $2^L|\Sigma|$ possible inputs. For each input, there are $2^L$ possible hash values. Thus, there are no more than $2^{L2^L|\Sigma|}$ compression functions. Moreover, there are no more than $2^L$ initial values $H_0$. Thus the number of iterated hash functions $|\mathcal{H}|$ is bounded by $2^{L(2^L|\Sigma|+1)}$:

$$|\mathcal{H}| \leq 2^{L(2^L|\Sigma|+1)}. \tag{2}$$

There are $|\Sigma|^n + |\Sigma|^{n-1} + \cdots + |\Sigma|$ possible non-empty strings.

Nguyen and Roscoe derived a bound which is particularly suited for values of $\varepsilon$ larger than $1/2^L$ [18]. Let $2^K$ be the number of hashable items. Pick any hash function. By the pigeon-hole principle, there must at least $2^K/2^L$ hashable items colliding to the same value. Apply any other hash function to these colliding items, there must be $2^K/2^{2L}$ items colliding on these two hash functions. By repeating this argument, Nguyen and Roscoe show that $\varepsilon$-almost universal hashing requires $\lceil K/L - 1 \rceil / \varepsilon$ hash functions [18].

**Corollary 1** (*Nguyen and Roscoe*)**.** *Given at least $2^K$ hashable items, then $\varepsilon$-almost universal hashing for $\varepsilon > 0$ requires at least $\lceil K/L - 1 \rceil / \varepsilon$ hash functions.*

**Proof.** We have $X = 2^r$ hash functions. Theorem 1 from [18] states that when $K$ is a multiple of $L$ then $r \geq \log(\varepsilon^{-1}(\frac{K}{L} - 1))$ and $r \geq \log(\varepsilon^{-1}\lfloor \frac{K}{L} \rfloor)$ otherwise. Their result may be rewritten as $r \geq \log(\varepsilon^{-1}(\lceil \frac{K}{L} - 1 \rceil))$, irrespective of the value of $K$. Thus, we have $X = 2^r \geq \lceil \frac{K}{L} - 1 \rceil / \varepsilon$. It proves the result. □

By combining this last corollary with our bound on the number of iterated hash functions (see Eq. (2)), we get the following bound on the universality of iterated hashing.

**Lemma 9.** *At best, iterated hashing might be $\varepsilon$-almost universal over the strings of length at most $L(\varepsilon 2^{L(2^{L+1}+1)} + 1)$ for some $\varepsilon < 1$.*

**Proof.** According to Corollary 1, we have that $|\mathcal{H}| \geq \lceil K/L - 1 \rceil / \varepsilon$. Solving for $K$ in this expression, we get $K \leq L(\varepsilon|\mathcal{H}| + 1)$. Meanwhile, for string of length at most $n$ over $|\Sigma|$ distinct characters, we have $|\Sigma|^n \leq 2^K$ or $n \log |\Sigma| \leq K$. Hence, by combining these two inequalities, we have $n \log |\Sigma| \leq K \leq L(\varepsilon|\mathcal{H}| + 1)$ or just

$$n \log |\Sigma| \leq L(\varepsilon|\mathcal{H}| + 1).$$

Moreover, we can bound the size of an iterated family as $|\mathcal{H}| \leq 2^{L(2^L|\Sigma|+1)}$ (see Eq. (2)). Hence, by substitution, we have

$$\begin{aligned} n \log |\Sigma| &\leq L(\varepsilon|\mathcal{H}| + 1) \\ &\leq L(\varepsilon 2^{L(2^L|\Sigma|+1)} + 1). \end{aligned}$$

Finally, we can solve for $n$ in this inequality. Thus – at best – iterated hashing might be almost universal over strings of length at most $L(\varepsilon 2^{L(2^L|\Sigma|+1)} + 1)/\log|\Sigma|$ where $|\Sigma| > 1$. This bound grows exponentially with $|\Sigma|$ which is misleading because universality over a large alphabet ($|\Sigma|$ large) implies universality over a smaller alphabet. Indeed, it is always possible to restrict the application of a universal family to strings using few characters, and this restriction may only increase the universality. Thus, it is preferable to set $|\Sigma| = 2$. (For $|\Sigma| = 1$, we get a weaker bound: we have that $2^K \geq n$ and $\mathcal{H} \leq 2^{L2^L}$ so that the bound becomes $n \leq 2^{L\varepsilon 2^{L2^L}+1}$.) □

For bounding universality, that is $1/2^L$-almost universality, it is preferable to use bound provided by Stinson [31] to get the following result.

**Lemma 10.** *At best, iterated hashing might be universal over the strings of length at most $2L + L2^{L+1}$. If the family is strongly universal, then it is limited to strings of at most $L + 2\log 2^L! - \log(2^L - 1) - 1$ characters.*

**Proof.** First, we consider universal hashing. By Eq. (2), we have that $2^{L(2^L|\Sigma|+1)} \geq |\mathcal{H}|$. Stinson [31] proved that the size of universal families must be at least as large as the number of hash values divided by the number of elements, thus we have $|\mathcal{H}| \geq |\Sigma|^n/2^L$. By combining these two inequalities, we get

$$2^{L(2^L|\Sigma|+1)} \geq |\mathcal{H}| \geq |\Sigma|^n/2^L$$

or

$$2^{L(2^L|\Sigma|+1)} \geq |\Sigma|^n/2^L.$$

Taking the logarithm on both sides, we get

$$n \leq \frac{L(2^L|\Sigma| + 2)}{\log|\Sigma|}.$$

The right-hand-side of this last inequality grows with $|\Sigma|$, but a family universal over a large alphabet must be universal over a smaller alphabet as well. Thus we set $|\Sigma| = 2$. This proves the first part of the lemma.

Consider strongly universal hashing. Recall that Stinson [31] proved that strongly universal families must have cardinality at least $1 + a(b - 1)$ where $a$ is the number of strings and $b$ is the number of hash values. Hence, we have that $|\mathcal{H}| \geq 1 + |\Sigma|^n(2^L - 1)$. As a consequence of Lemma 4, strongly universal hash families must have permuting compression functions. There are $(2^L!)^{|\Sigma|}$ such functions, and $2^L$ possible initial values for a total of at most $2^L \times 2^L!^{|\Sigma|}$ hash functions. Hence, we have $2^L \times 2^L!^{|\Sigma|} \geq |\mathcal{H}|$. By combining these inequalities, we get

$$2^L \times 2^L!^{|\Sigma|} \geq 1 + |\Sigma|^n(2^L - 1).$$

We can drop the constant term 1 from the right to get

$$|\Sigma|^n < \frac{2^L \times 2^L!^{|\Sigma|}}{2^L - 1}.$$

As before, we can set $|\Sigma| = 2$ to get

$$n < L + 2\log 2^L! - \log(2^L - 1).$$

This concludes the proof. □

**Table 5**
Comparison of the bounds on universality from Lemma 10 and Proposition 4.

| $L$ | Lemma 10 | | Proposition 4 |
|---|---|---|---|
| | Universality | Strong univer. | Universality |
| 2 | 20 | 8 | 5 |
| 4 | 136 | 87 | 17 |
| 8 | 4 112 | 3 366 | 257 |
| 16 | 2 097 184 | 1 908 072 | 65 537 |

## 7. Limitations of iterated hashing over long strings

To characterize the limitations of iterated hashing—irrespective of the family size, we want to compute a bound on the string length given a desired bound $\varepsilon$ on the collision probability.

Let $s_{r,\mathtt{a}}$ be the unary string made of the character a repeated $r$ times. For example, we have $s_{3,\mathtt{a}} = \mathtt{aaa}$. Because we have at most $2^L$ distinct hash values, we have that $h(s_{2^L+1,\mathtt{a}})$ must be equal to $h(s_{r,\mathtt{a}})$ for some $r \in \{1, \ldots, 2^L\}$. Hence, we have the following lemma.

**Lemma 11.** *For any iterated hash function $h$ and any character* a*, the values $h(s_{r,\mathtt{a}})$ are cyclic over $r \geq 1$ with a period $T \in \{1, 2, \ldots, 2^L\}$ except maybe for the first $2^L - T$ hash values.*

**Proof.** In the $2^L + 1$ hash values $h(s_{r,\mathtt{a}})$ for $r \in \{1, 2, \ldots, 2^L, 2^L + 1\}$, one value must be repeated because there are at most $2^L$ distinct hash values. Write $h(s_{r_1,\mathtt{a}}) = h(s_{r_2,\mathtt{a}})$, then by Proposition 1, $h(s_{r_1+i,\mathtt{a}}) = h(s_{r_2+i,\mathtt{a}})$ for any non-zero integer $i$. Without loss of generality, assume $r_2 > r_1$. This proves that $h(s_{r_1+x,\mathtt{a}})$ is cyclic in $x$ with period at most $T = |r_1 - r_2|$. We see that $1 \leq T \leq 2^L$. Only the $h(s_{i,\mathtt{a}})$ for $i = 1, 2, \ldots, r_1 - 1$ are excluded from our analysis. This concludes the proof. □

Let $LCM_k \equiv LCM(\{1, 2, \ldots, k\})$ be the least common multiple of the integers from 1 to $k$, inclusively. For example, we have $LCM_2 = 2$, $LCM_4 = 12$, $LCM_8 = 840$. By definition, for any $T \in \{1, 2, \ldots, k\}$, we have that $T$ divides $LCM_k$. Thus, the strings $s_{2^L,\mathtt{a}}$ and $s_{2^L+LCM_{2^L},\mathtt{a}}$ collide with probability one under iterated hashing by Lemma 11. Using a generalized argument, we have the following proposition.

**Proposition 4.** *We have the following results concerning iterated hashing over variable-length strings:*

- *almost universality over strings of length up to $2^L + LCM_{2^L}$ is impossible;*
- *universality over strings of length up to $2^L + 2$ is impossible;*
- *for $1/2^L < \varepsilon < 1$ such that $1/\varepsilon$ is not an integer, $\varepsilon$-almost universality over strings of length at most $2^L + LCM_{2^L+1-\lfloor 1/\varepsilon \rfloor}$ is impossible.*

**Proof.** Since the values $h(s_{r+2^L-1,\mathtt{a}})$ must have period $T \in \{1, \ldots, 2^L\}$ as functions of $r$, and $T$ must divide $LCM_{2^L}$, we have that $h(s_{2^L,\mathtt{a}}) = h(s_{2^L+LCM_{2^L},\mathtt{a}})$ for all iterated hash functions $h$ (see Lemma 11). This proves the first result.

We prove the last result. Suppose that hashing is $\varepsilon$-almost universal. Then the probability that $h(s_{r+2^L-1,\mathtt{a}})$ is cyclic in $r$ with period $T$ is bounded by $\varepsilon$: $P(\text{period}(h) = T) \leq \varepsilon$. Thus, we have $P(\text{period}(h) \in \{2^L - j + 1, 2^L - j + 2, \ldots, 2^L\}) \leq j\varepsilon$ for any integer $j$ between 1 and $2^L$. Because $P(\text{period}(h) \in \{1, 2, \ldots, 2^L\}) = 1$, we have $P(\text{period}(h) \leq 2^L - j) \geq 1 - j\varepsilon$. Setting $j = \lfloor 1/\varepsilon \rfloor - 1$, we have that $P(\text{period}(h) \leq 2^L - j) \geq 1 - (\lfloor 1/\varepsilon \rfloor - 1)\varepsilon = \varepsilon - \lfloor 1/\varepsilon \rfloor \varepsilon > \varepsilon$. Thus, the probability $P(h(s_{2^L,\mathtt{a}}) = h(s_{LCM_{2^L+1-\lfloor 1/\varepsilon \rfloor}+2^L,\mathtt{a}})) > \varepsilon$ concludes the proof of the last item.

The second result follows because universality implies $1/2^L+\delta$-almost universality for all $\delta > 0$. We can find $\delta$ sufficiently small, such that $1/\varepsilon$ is not an integer, and such that $\lfloor 1/\varepsilon \rfloor = 2^L - 1$. Thus, we have that universality over strings of length at most $2^L + LCM_2 = 2^L + 2$ is impossible. This concludes the proof. □

We have that $LCM_{2^L}$ divides $2^L!$ so $LCM_{2^L} \leq 2^L!$; moreover, by a standard identity $2^L! < (2^L)^{2^L} = 2^{L2^L}$. Hence, the bound on almost universality from this last proposition is preferable to the cardinality-based bound (see Lemma 9). Similarly, we compare the bounds on universality in Table 5: the new bound of $2^L + 1$ characters is much smaller.

At least for unary strings, the next lemma shows that the almost universality bound of Proposition 4 is tight (up to one character).

**Lemma 12.** *There exists an almost universal iterated family over unary strings of length at most $2^L + LCM_{2^L} - 2$.*

**Proof.** For $T \in \{1, 2, \ldots, 2^L\}$, define

$$h_T(s_{r,\mathtt{a}}) = \begin{cases} r & \text{if } 0 \leq r < 2^L \\ 2^L - T - (r - 2^L \bmod T) & \text{otherwise} \end{cases}.$$

Effectively, $h_T$ goes from 0 to $2^L - 1$ for strings of length 0 to $2^L - 1$, and then it becomes cyclic with period $T$ (see Fig. 3). This family is iterated.

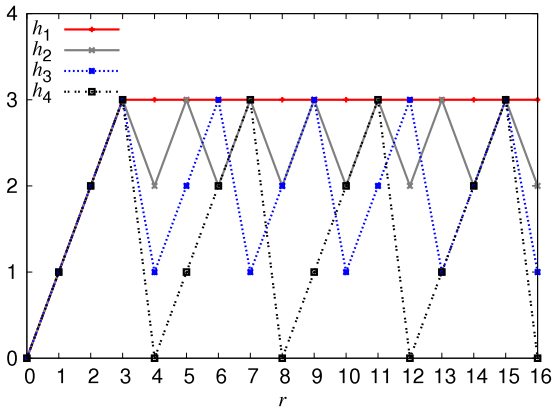


**Fig. 3.** Functions $r \to h_T(s_{r,\mathrm{a}})$ for $L = 2$ and $T = 1, 2, 3, 4$.

We want to show that there is no pair of strings $s_{r,\mathrm{a}}, s_{r',\mathrm{a}}$ for $r, r' \leq LCM_{2^L} + 2^L - 1$ such that $h_T(s_{r,\mathrm{a}}) = h_T(s_{r',\mathrm{a}})$ for all $T \in \{1, 2, \ldots, 2^L\}$. Suppose that it is false. It cannot happen if $r, r' < 2^L$ since $h_T(s_{r,\mathrm{a}}) = h_T(s_{r',\mathrm{a}})$ would imply $r = r'$. Suppose that $h_T(s_{r,\mathrm{a}}) = h_T(s_{r',\mathrm{a}})$ for all $T \in \{1, 2, \ldots, 2^L\}$, and for some $2^L \leq r \leq LCM_{2^L} + 2^L - 2$ and some $r' < 2^L$. Then $h_T(s_{r',\mathrm{a}}) = r' - 1$. This would imply that $h_T(s_{r,\mathrm{a}})$ is independent of $T$ which is not possible for $2^L < r < LCM_{2^L} + 2^L$ because $h_T(s_{r,\mathrm{a}})$ is cyclic with period $T$. Similarly, for $2^L \leq r, r' < LCM_{2^L} + 2^L - 1$, the equality $h_T(s_{r,\mathrm{a}}) = h_T(s_{r',\mathrm{a}})$ is possible only when $T \in \{1, 2, \ldots, 2^L\}$ divides $|r - r'|$. But since $|r - r'| < LCM_{2^L}$, this is not possible for all $T \leq 2^L$.

The result is shown. □

Finally, we show that the universality bound of Proposition 4 is tight for unary strings (up to two characters). To prove the result, we build a perfect (collision-free) hash function over unary strings of length at most $2^L$. Consider strings made of the character a. Let $\pi$ be a cycle of length $2^L$ over the integers $\{1, 2, \ldots, 2^L\}$. For example, let $\pi$ be the permutation that takes $k$ to $k + 1$ for $k < 2^L$ and $2^L$ to 1. We choose the hash function $h(s_{r,\mathrm{a}}) = \pi^r 1$. A collision between any two strings of length at most $2^L$ implies that $\pi^l H_0 = H_0$ for some $l < 2^L$ which is impossible because $\pi$ is a cycle of length $2^L$. Thus, no two unary strings of length at most $2^L$ may collide under this hash function.

## 8. Conclusion

We have shown that iterated hashing can be pairwise independent over short strings. Moreover, iterated hashing can be almost universal over strings longer than the number of hash values. Motivated by this result, we have derived bounds on the universality of iterated hashing. We can construct large iterated hashing families: this might suggest that a very high degree of universality is possible. Alas we have shown that this expectation would be misguided

We have identified two open problems which we find interesting. On the one hand, we lack a bound on the universality of iterated hashing given the size of the family. Our bounds assume arbitrarily large families. On the other hand, we are still missing provably optimal iterated families. For example, is it possible to construct a family which is pairwise independent for strings longer than $L$ for some values of $L > 1$? Future work might consider the specific limitations of other hashing strategies [31,29].

## Acknowledgment

This work is supported by NSERC grant 261437.

## Appendix. Other popular iterated hash functions

For completeness, we review some popular iterated hash families and functions. We show that many of these families have permuting compression functions.

### *A.1. Hashing by random irreducible polynomials*

Instead of using a fixed irreducible polynomial, as in CWPoly, we can pick the irreducible polynomial at random [10, 30] (henceforth Division). Considering $L$-bit characters as polynomials having binary coefficients (GF(2)[$x$]), we use the compression function $F(y, c) = yx^L + c$. As in CWPoly, we specify an initial value of 1. Thus, given a string $s$, the hash value is $x^{nL} + s_1 x^{(n-1)L} + \cdots + s_n \bmod p(x)$. Consider two strings of length at most $n$. The equation $h(s) = h(s')$ is true in GF(2)[$x$]/$p(x)$ only if the non-zero polynomial of degree at most $nL$ formed by $h(s) - h(s')$ is divisible by $p(x)$. The polynomials of degree

$nL$ have at most $n$ irreducible factors of degree $L$. Meanwhile, there are at least $(2^L - 2^{L/2+1})/L$ irreducible polynomials $p(x)$ of degree $L$ [22]. Thus, the probability of a collision is no larger than $\frac{nL}{2^L - 2^{L/2+1}}$.

We can extend this analysis to show almost XOR universality with the same bound ($\frac{nL}{2^L - 2^{L/2+1}}$). Pick any value $y$, then the probability $P(h(s) \oplus h(s') = y)$ is given by the probability that $h(s) + h(s') - y = 0$ in $\mathrm{GF}(2)[x]/p(x)$. The polynomial $h(s) + h(s') - y$ in $\mathrm{GF}(2)[x]$ has degree at most $nL$ but at least $L$, and the result follows.

The compression function of Division is (strongly) permuting: $F(y, c) = F(y', c)$ implies $x^L y = x^L y' \mod p(x)$ which implies $y = y'$. Moreover, if we forbid the character value zero at the beginning of strings, and pick the initial value randomly, we have that Division is uniform and thus, $\frac{nL}{2^L - 2^{L/2+1}}$-almost strongly universal.

We might be able to compute Division faster than CWPoly. However, selecting a random irreducible polynomial might be slow. To cope with this problem, Shoup introduced a generalized Division [30] with compression function $F(y, c) = yx^{L/k} + c$ and $p(x)$ chosen as a monic irreducible polynomial of degree $L/k$.

### A.2. Bernstein hashing

Bernstein proposed a computationally efficient compression function [2]: $F(y, c) = ((y \ll l) + y) \oplus c$ where $y \ll l$ is the left shift by $l$ bits. For all $l > 0$, this compression function is strongly permuting. Hence, given randomly chosen initial values, we have uniform hashing.

### A.3. Fowler–Noll–Vo hashing

There are two types of Fowler–Noll–Vo hash functions [19]. The FNV-1 compression functions takes the form $F(y, c) = (yp) \oplus c$ where $p$ is a prime number. It is a generalization of Bernstein hashing. The FNV-1a compression functions are of the form $F(y, c) = (y \oplus c)p$ for some prime $p$. Both FNV-1 and FNV-1a are strongly permuting.

### A.4. SAX and SXX

The shift-add-xor (SAX) scheme [25] is defined by the compression function $F(y, c) = y \oplus ((y \ll l) + (y \gg r) + c)$ where $y \gg r$ is the right shift by $r$ bits. For $L = 32$ and 7-bit characters, Ramakrishna and Zobel [25] reported that SAX is *empirically universal* for $4 \leq l \leq 7$ and $1 \leq r \leq 3$, and a randomly chosen 32-bit initial value. They found that the alternative, shift-xor-xor (SXX), $F(y, c) = y \oplus ((y \ll l) \oplus (y \gg r) \oplus c)$, is not competitive.

### A.5. String hashing functions in common programming languages

Strings are commonly used as keys in hash tables. Thus, most programming languages include string hashing functions. We consider C++ and Java.

ISO added support for hash tables to the C++ language (`unordered_map`) [35]. Implementations of the language are required to provide a string hashing function, but the exact function is unspecified. However, a popular compiler (GNU GCC, version 4.1.1) implemented it as an iterated hash function with the compression function $F(y, c) = 5y + c \mod 2^{32}$ and an initial value of zero. For example, the hash value of the one-character string `z` is 122, the decimal value corresponding to the character `z`.

The Java `String` class has a specified `hashCode` method. As of version 1.3 of the language, it is an iterated hash function with compression function $F(y, c) = 31y + c$ – using `int` arithmetic – and an initial value of zero. Because Java lacks unsigned integers as a native type, the hash value of a sufficiently long string (e.g., `zzzzzz`) can be a negative integer. (Java uses the Two's complement binary representation, so that signed integers are interchangeable with unsigned integers as long as we only use addition, subtraction and multiplication.)

### A.6. PowerOfTwo *hashing*

In light of the hash functions used in Java and C++, consider the family given by the compression function $F(y, c) = By + c \mod 2^L$ (henceforth PowerOfTwo). If the initial value is zero, then $h(00) = h(0) = 0$, but we can fix this problem by using a non-zero initial value. However, suppose that $B$ is even and consider any two strings $s$ and $s'$ of length greater than $L$ and differing only in the first character, then $h(s) = h(s')$ because $B^L \mod 2^L = 0$. Thus – unsurprisingly – both the C++ and Java implementations set $B$ to an odd integer.

Suppose that $B$ is odd. The compression function is then strongly permuting. Thus, by choosing the initial value at random, we have uniform hashing.

When $B$ is odd, we have that $B + 1$ is even, so that $(B + 1)^L \mod 2^L = 0$. By the binomial theorem, we have that $0 = (B + 1)^L \bmod 2^L = \sum_{k=0}^{L} B^k (\binom{L}{k} \mod 2^L) \bmod 2^L$. Thus – irrespective of the initial value – the two strings of length $L + 1$ given by the characters $\binom{L}{k} \mod 2^L$ for $k = 0, 1, \ldots, L$ and $00 \cdots 0$ collide when $B$ is odd. By this construction, PowerOfTwo cannot be almost universal unless we limit the length of the strings to at most $L$ characters.

## References


[1] K. Acken, M. Irwin, R. Owens, Power comparisons for barrel shifters, in: ISLPED'96, 1996, pp. 209–212.
[2] D.J. Bernstein, CDB-Constant Database, Checked 2011-09-01. http://cr.yp.to/cdb.html.
[3] J. Byers, J. Considine, M. Mitzenmacher, Simple load balancing for distributed hash tables, Lecture Notes in Computer Science (2003) 80–87.
[4] L. Carter, M.N. Wegman, Universal classes of hash functions, Journal of Computer and System Sciences 18 (2) (1979) 143–154.
[5] J.D. Cohen, Recursive hashing functions for *n*-grams, ACM Transactions on Information Systems 15 (3) (1997) 291–320.
[6] M. Dietzfelbinger, Universal hashing and *k*-wise independent random variables via integer arithmetic without primes, in: STACS'96, 1996, pp. 569–580.
[7] M. Fürer, Faster integer multiplication, in: STOC'07, 2007, pp. 57–66.
[8] S. Gueron, M.E. Kounavis, Intel carry-less multiplication instruction and its usage for computing the GCM mode—Rev. 2, White Paper, January 2010.
[9] D.E. Knuth, Sorting and Searching, in: Art of Computer Programming, vol. 3, Addison-Wesley, 1997.
[10] H. Krawczyk, LFSR-based hashing and authentication, Lecture Notes in Computer Science 839 (1994) 129–139.
[11] H. Krawczyk, New hash functions for message authentication, Lecture Notes in Computer Science 921 (1995) 301–310.
[12] T. Krovetz, P. Rogaway, Fast universal hashing with small keys and no preprocessing: the PolyR construction, Lecture Notes in Computer Science 2015 (2001) 73–89.
[13] K. Kukich, Techniques for automatically correcting words in text, ACM Computing Surveys 24 (4) (1992) 377–439.
[14] D. Lemire, O. Kaser, Recursive *n*-gram hashing is pairwise independent, at best, Computer Speech and Language 24 (4) (2010) 698–710.
[15] M. Liskov, Constructing an ideal hash function from weak ideal compression functions, Lecture Notes in Computer Science 4356 (2007) 358.
[16] R.C. Merkle, Secrecy, authentication, and public key systems, Ph.D. Thesis, Stanford University, Stanford, CA, USA, 1979.
[17] K.S. Ng, L.M. Cheng, C.H. Wong, Dynamic word based text compression, in: ICDAR'97, 1997.
[18] L.H. Nguyen, A.W. Roscoe, New combinatorial bounds for universal hash functions, Tech. Rep. 2009/153, Cryptology ePrint Archive, 2009. http://eprint.iacr.org/.
[19] L.C. Noll, Fowler/Noll/Vo Hash, Checked 2011-09-01, 2001. http://www.isthe.com/chongo/tech/comp/fnv/.
[20] R. Pagh, F. Rodler, Cuckoo hashing, Journal of Algorithms 51 (2) (2004) 122–144.
[21] P.K. Pearson, Fast hashing of variable-length text strings, Communications of the ACM 33 (6) (1990) 677–680.
[22] P. Piret, On the number of divisors of a polynomial over GF(2), in: Applied Algebra, Algorithmics and Error-Correcting Codes, Springer, 1986, pp. 161–168.
[23] B. Preneel, R. Govaerts, J. Vandewalle, Hash functions based on block ciphers: a synthetic approach, Lecture Notes in Computer Science 773 (1994) 368–378.
[24] M.V. Ramakrishna, Practical performance of bloom filters and parallel free-text searching, Communications of the ACM 32 (10) (1989) 1237–1239.
[25] M.V. Ramakrishna, J. Zobel, Performance in practice of string hashing functions, in: Proc. Int. Conf. on Database Systems for Advanced Applications, 1997.
[26] P. Rogaway, Bucket hashing and its application to fast message authentication, Journal of Cryptology 12 (2) (1999) 91–115.
[27] F. Ruskey, The (combinatorial) object server, Checked 2011-09-01, 2006. http://www.theory.cs.uvic.ca/~cos/cos.html.
[28] P. Sarkar, A new universal hash function and other cryptographic algorithms suitable for resource constrained devices, Tech. Rep. 2008/216, Cryptology ePrint Archive, 2008. http://eprint.iacr.org/.
[29] P. Sarkar, A trade-off between collision probability and key size in universal hashing using polynomials, Designs, Codes and Cryptography 58 (2011) 271–278.
[30] V. Shoup, On fast and provably secure message authentication based on universal hashing, Lecture Notes in Computer Science 1109 (1996) 313–328.
[31] D.R. Stinson, Universal hashing and authentication codes, Designs, Codes and Cryptography 4 (4) (1994) 369–380.
[32] M. Thorup, Even strongly universal hashing is pretty fast, in: SODA'00, 2000, pp. 496–497.
[33] M. Thorup, String hashing for linear probing, in: SODA'09, 2009, pp. 655–664.
[34] M. Thorup, Y. Zhang, Tabulation based 4-universal hashing with applications to second moment estimation, in: SODA'04, 2004, pp. 615–624.
[35] Working Draft, Standard for programming language C++, Checked 2011-09-01, 2009. http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2009/n3000.pdf.
[36] A.L. Zobrist, A new hashing method with application for game playing, Tech. Rep. 88, Computer Sciences Department, University of Wisconsin, Checked 2011-09-01, 1970. http://www.cs.wisc.edu/techreports/viewreport.php?report=88.
[37] A.L. Zobrist, A new hashing method with application for game playing, ICCA Journal 13 (2) (1990) 69–73.